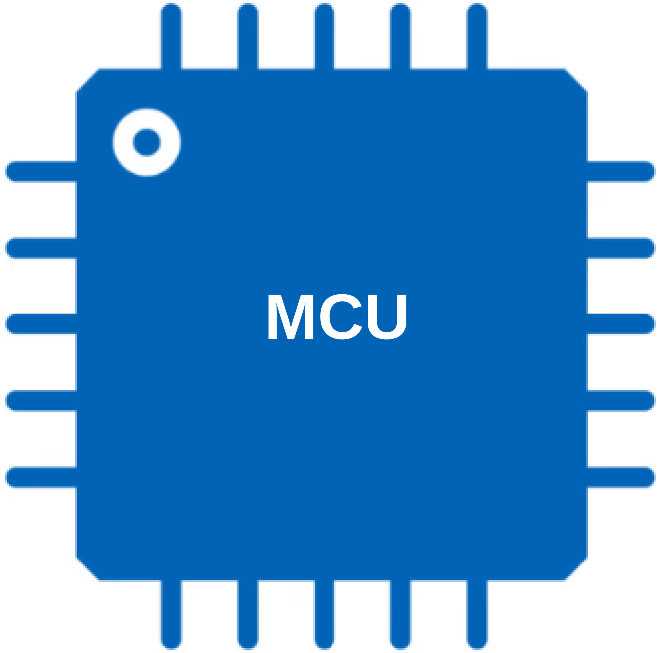
Firmware Development Training
GPIO, CLOCK, TIMERS & COUNTER, INTERRUPTS

26/03/2024 | Rooban G | SGC-DIN-RD

Getting stared with MCU based product design

Input

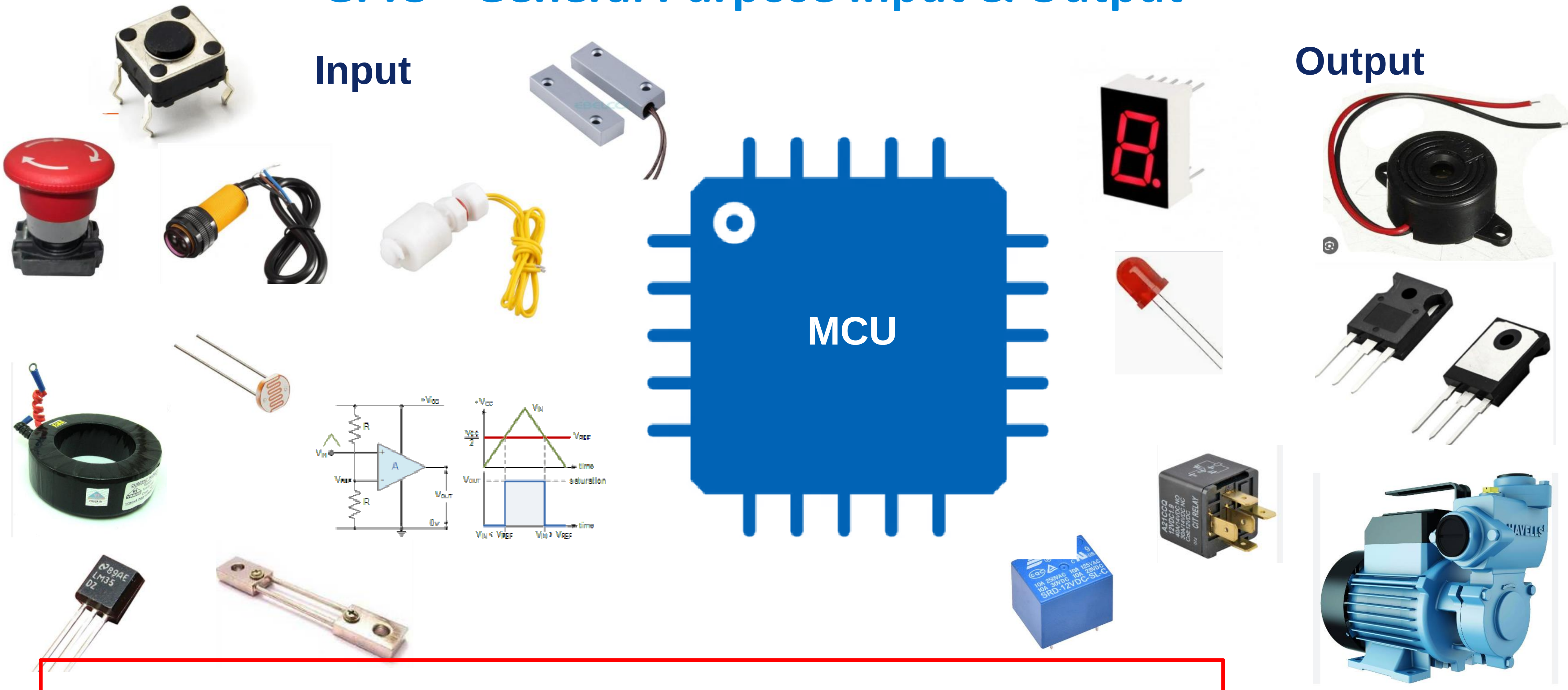
- Input from Human (HMI)
- From Physical parameters through sensors
- External Device communication



Output

- Indication to Human (HMI)
- Actuators
- External Device communication

GPIO - General Purpose Input & Output



Input

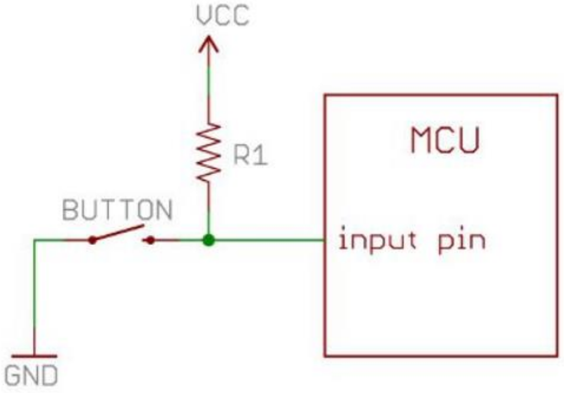
MCU

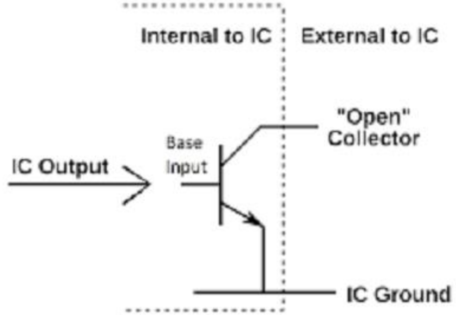
Output

GPIO deals with only Boolean signal => True or False, ON or OFF, 1 or 0, High or Low

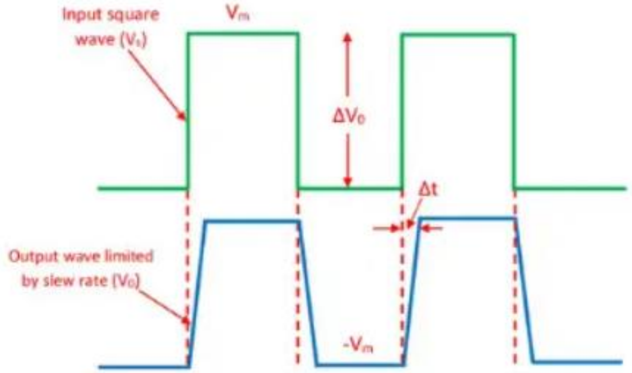
Configuration of GPIO, the SFRs

- Mostly all the PINs of an MCU (except power, clock, spl), can be configured as GPIO
- GPIOs are Port Mapped (PORT A, PORT B..)
- The configurations related to GPIO include
  - Direction
  - Pulls
  - Drive strength
  - Push pull vs OD
  - Interrupts
  - Slew rate,,
  - Multiplexer selection

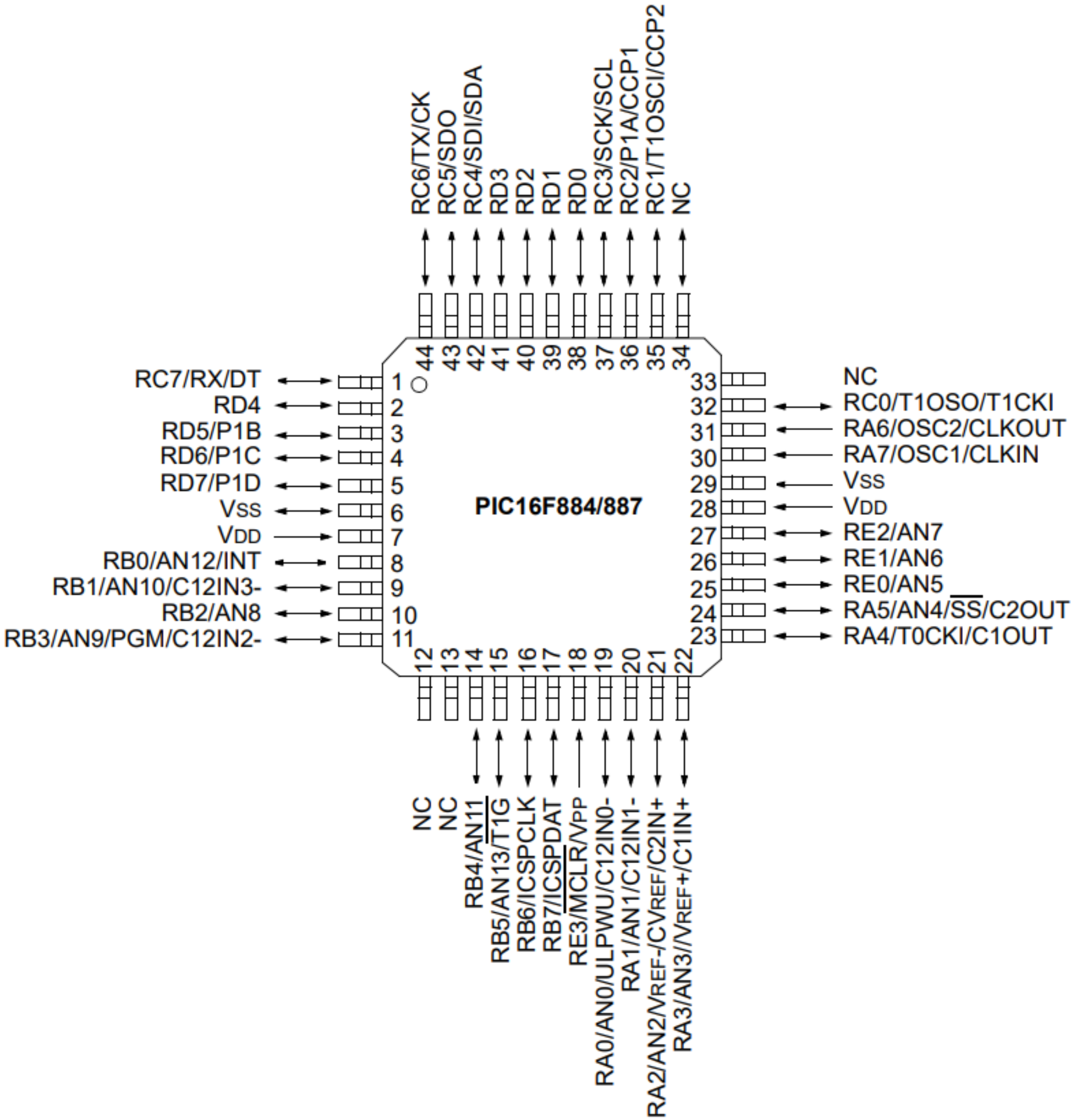




- The configurations are done by altering the values (bits) in SFRs

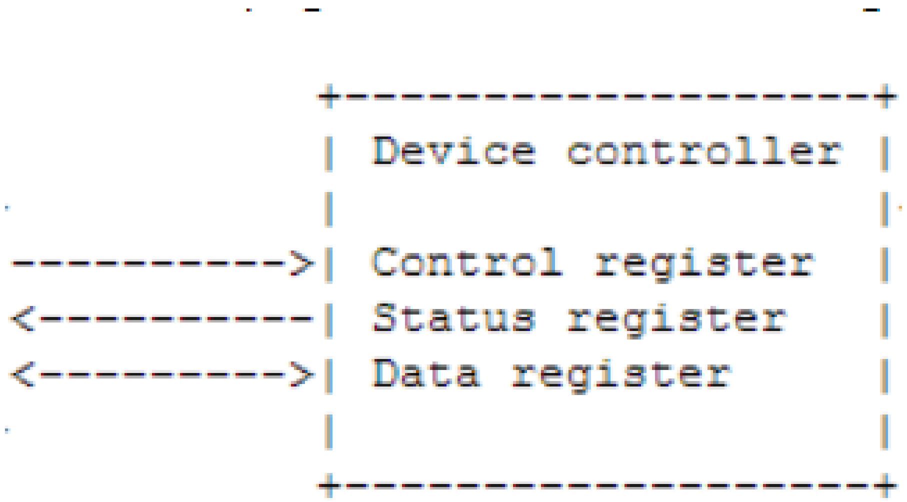

Input square wave (V_i)
V_m
ΔV_0
Δt
Output wave limited by slew rate (V_0)
$-V_m$

GPIO in PIC16F887 MCU



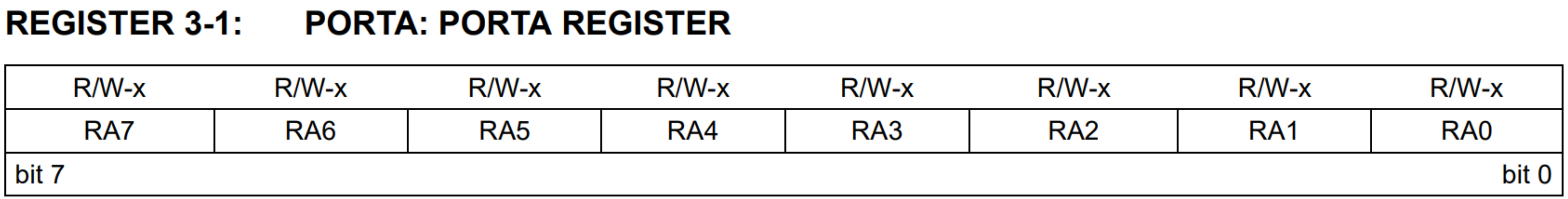
SFRs

- Data register
- Control register
- Status register



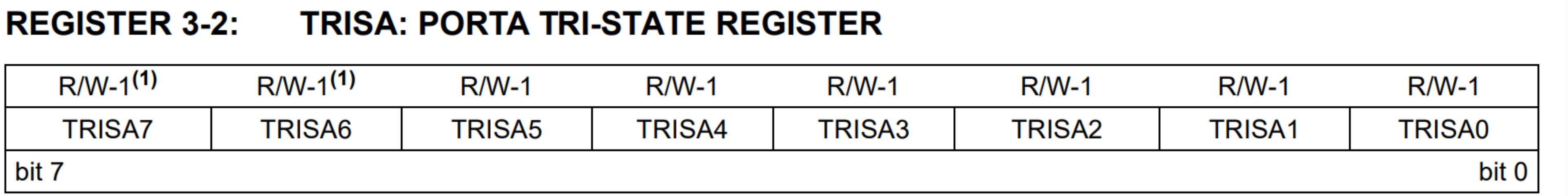
```
            +--------------------+
            | Device controller  |
            |                    |
----------->| Control register   |
<-----------| Status register    |
<---------->| Data register      |
            |                    |
            +--------------------+
```

Port A GPIO registers



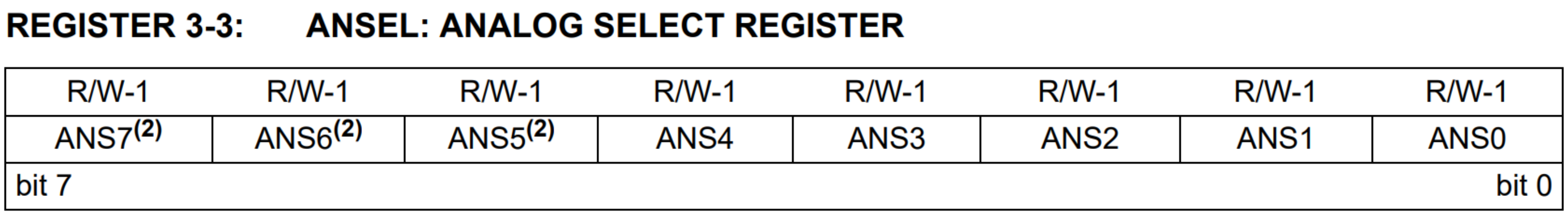
REGISTER 3-1: PORTA: PORTA REGISTER

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
bit 7							bit 0



REGISTER 3-2: TRISA: PORTA TRI-STATE REGISTER

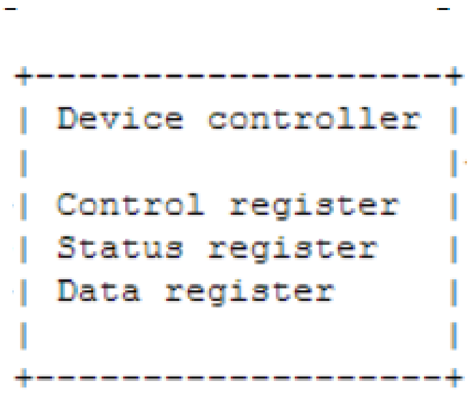
R/W-1[1]	R/W-1[1]	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
bit 7							bit 0



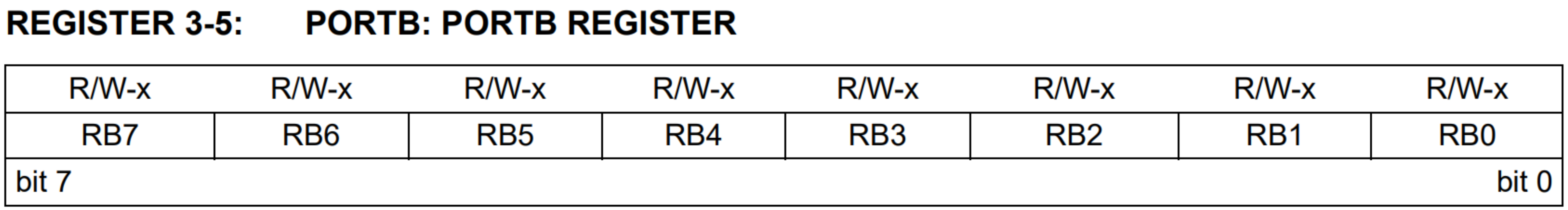
REGISTER 3-3: ANSEL: ANALOG SELECT REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
ANS7[2]	ANS6[2]	ANS5[2]	ANS4	ANS3	ANS2	ANS1	ANS0
bit 7							bit 0

Port B – GPIO registers

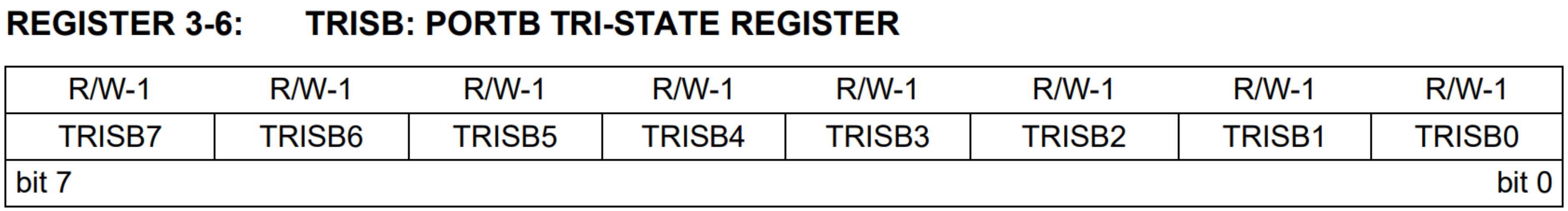


```
+--------------------+
| Device controller  |
|                    |
| Control register   |
| Status register    |
| Data register      |
|                    |
+--------------------+
```



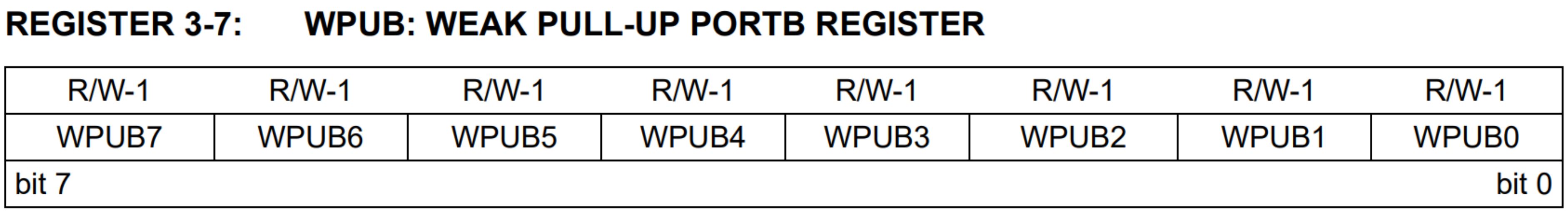
REGISTER 3-5: PORTB: PORTB REGISTER

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
bit 7							bit 0



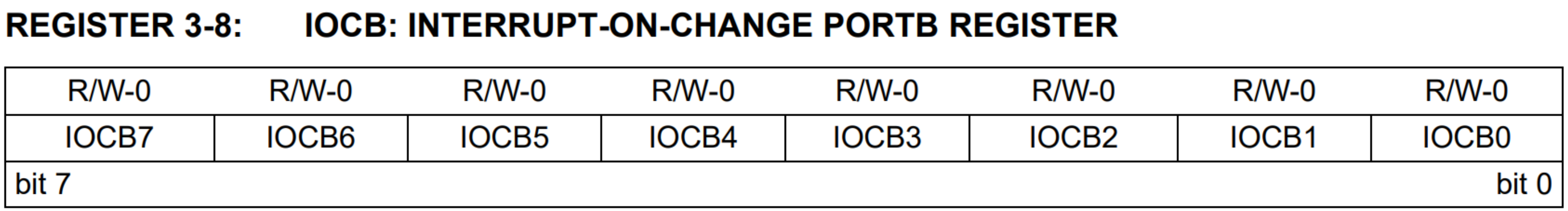
REGISTER 3-6: TRISB: PORTB TRI-STATE REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
bit 7							bit 0



REGISTER 3-7: WPUB: WEAK PULL-UP PORTB REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
bit 7							bit 0

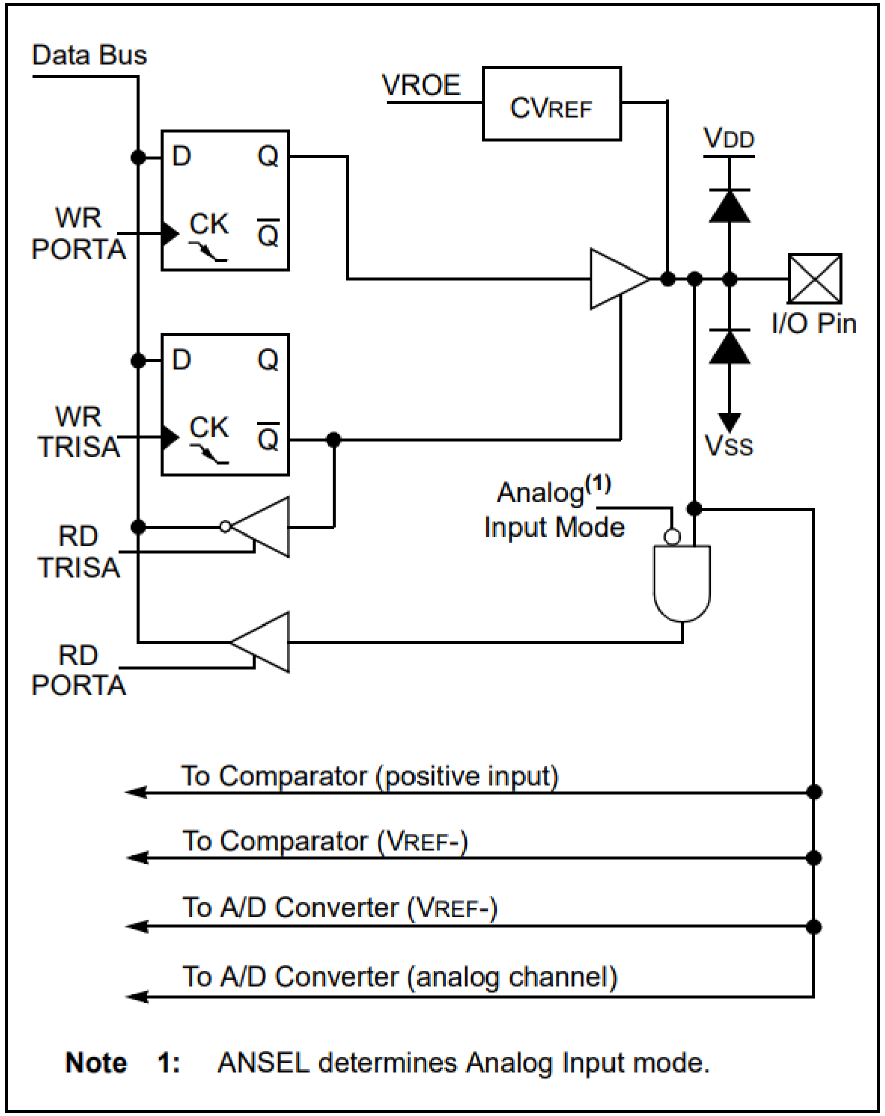


REGISTER 3-8: IOCB: INTERRUPT-ON-CHANGE PORTB REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
bit 7							bit 0

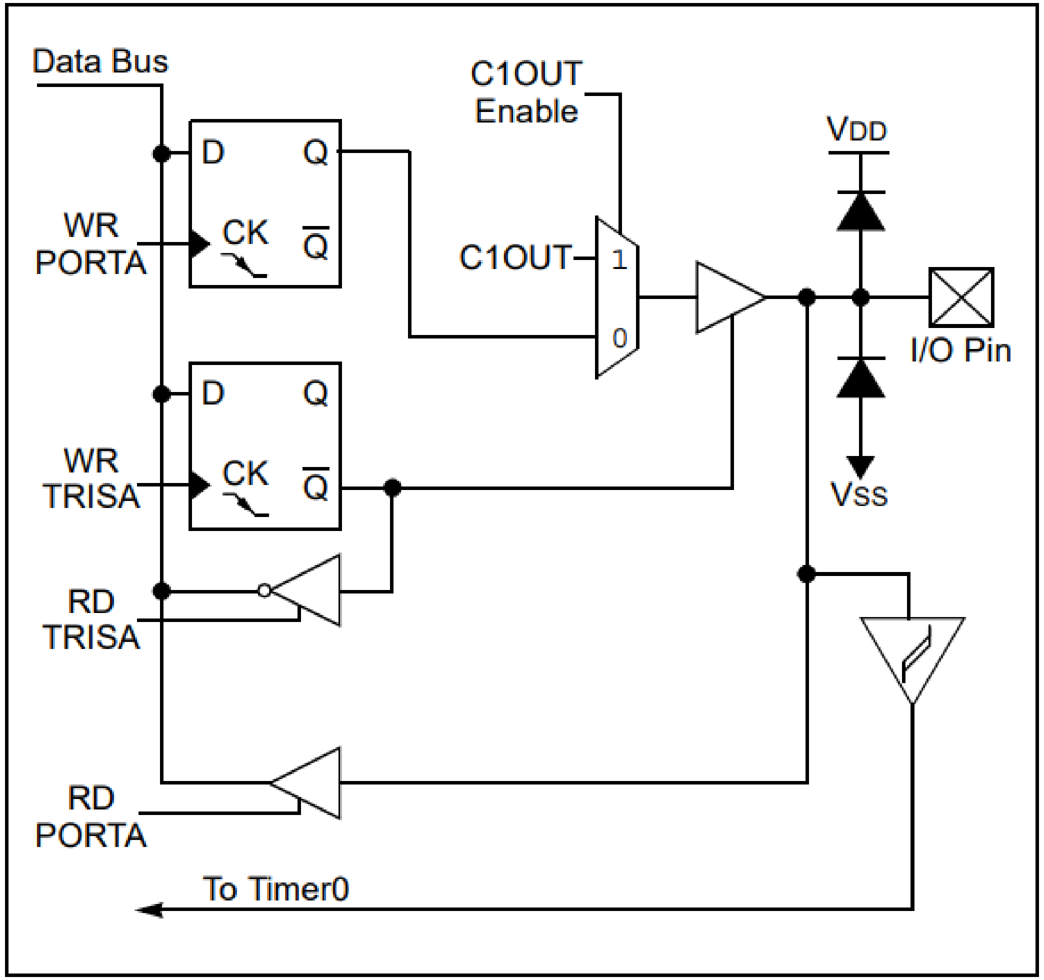
Port A – PIN 1 internal representation

FIGURE 3-3: BLOCK DIAGRAM OF RA2

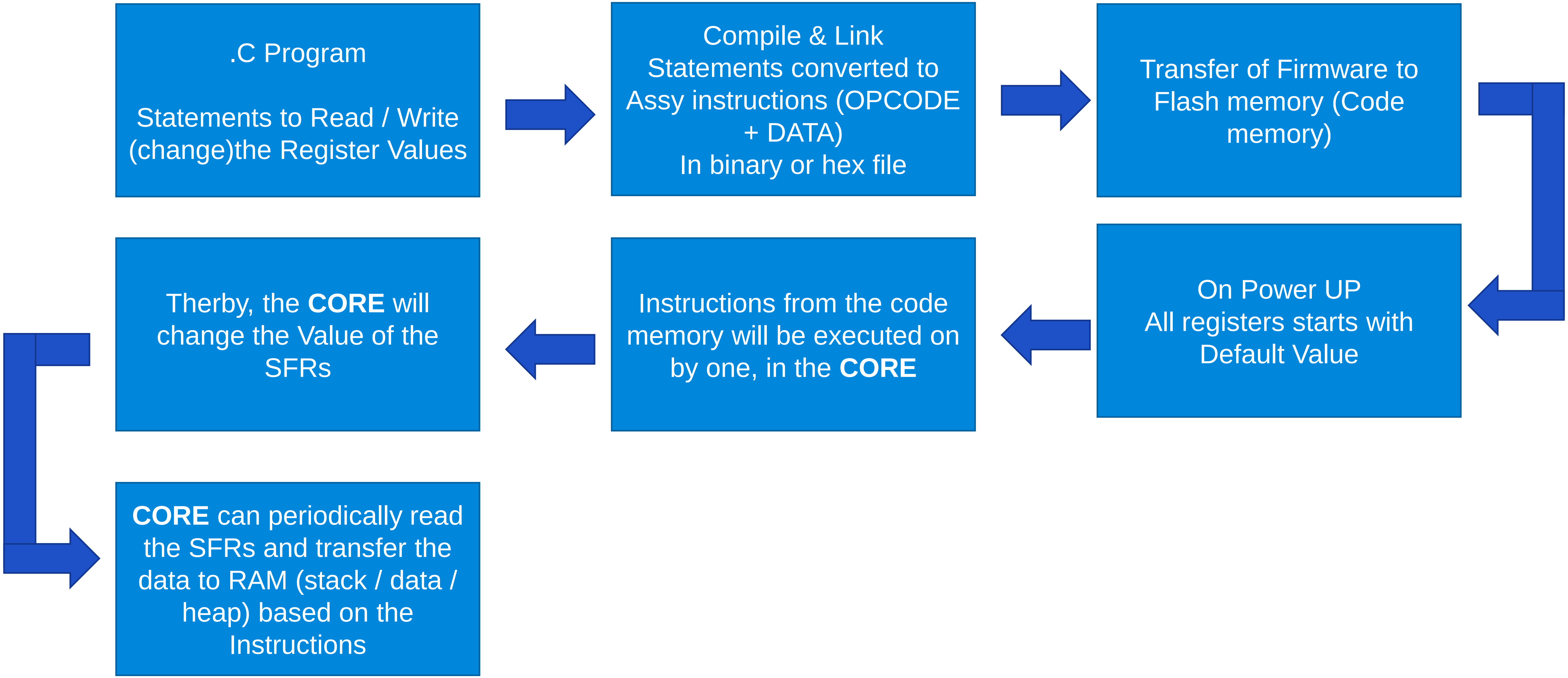


Note 1: ANSEL determines Analog Input mode.

FIGURE 3-5: BLOCK DIAGRAM OF RA4



How to change these registers?



.C Program

Statements to Read / Write (change)the Register Values

Compile & Link
Statements converted to Assy instructions (OPCODE + DATA)
In binary or hex file

Transfer of Firmware to Flash memory (Code memory)

On Power UP
All registers starts with Default Value

Instructions from the code memory will be executed on by one, in the **CORE**

Therby, the **CORE** will change the Value of the SFRs

CORE can periodically read the SFRs and transfer the data to RAM (stack / data / heap) based on the Instructions

Program Flow

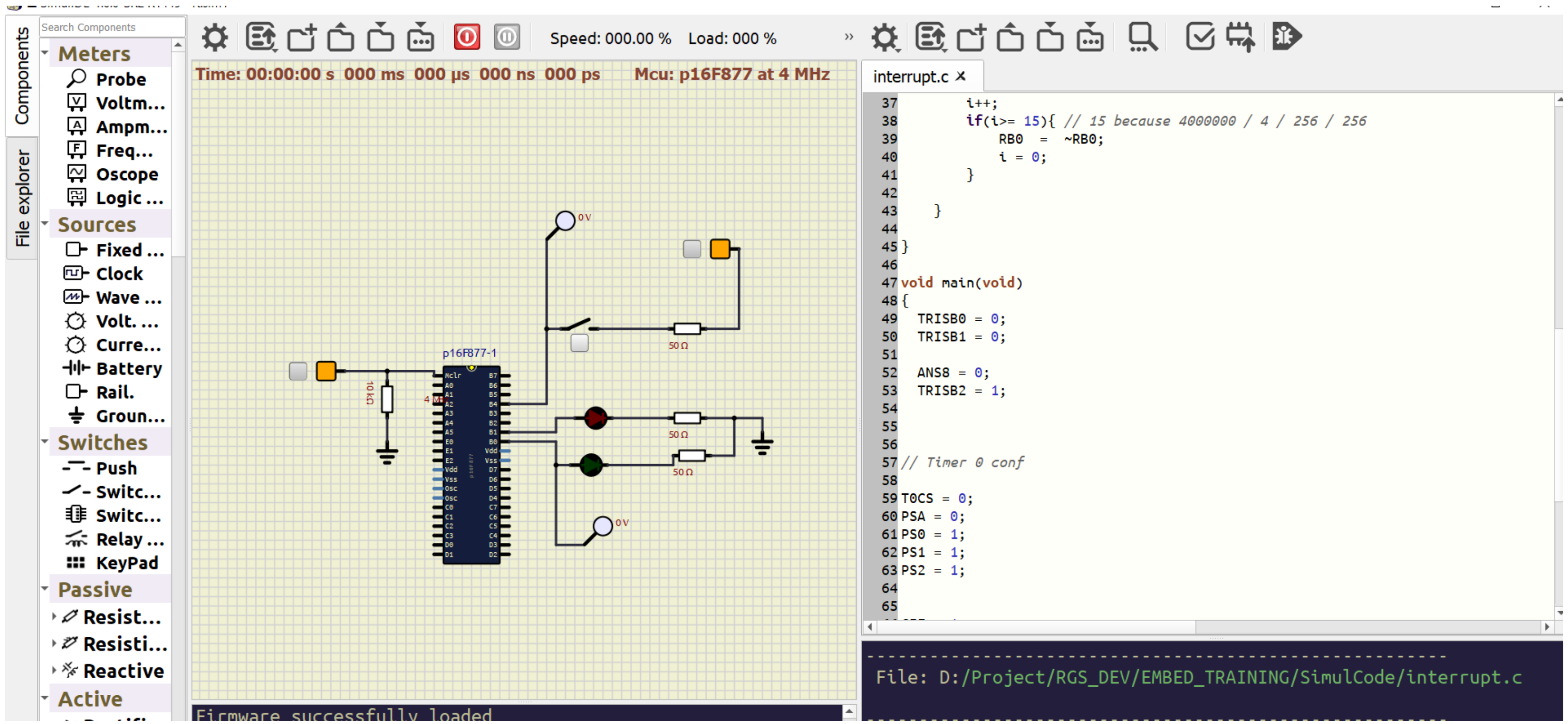
- Prepare the Toolchain / compiler (libraries)
  - One time execution Code (the configuration, mostly)
  - Continuous execution Code (the data collection / updating)

```
setup()
{

}



loop()
{

}
```

```
void main()
{

    //one time configuration area

while(1)
    {

    // continuous run area

    }
}
```

Hands – on programming & simulation

- Simulation with tool
  - SimulIDE is a electronic simulation tool, which supports Microcontroller simulations and programming
  - We also need to install the appropriate compilers for the controller which we are going to simulate,
    - (Eg Microchip’s XC8 compiler is needed for simulating PIC16F microcontrollers)



Speed: 000.00 % Load: 000 %

Time: 00:00:00 s 000 ms 000 µs 000 ns 000 ps   Mcu: p16F877 at 4 MHz

```
        i++;
        if(i>= 15){ // 15 because 4000000 / 4 / 256 / 256
            RB0  =  ~RB0;
            i = 0;
        }

    }

}

void main(void)
{
  TRISB0 = 0;
  TRISB1 = 0;

  ANS8 = 0;
  TRISB2 = 1;



// Timer 0 conf

T0CS = 0;
PSA = 0;
PS0 = 1;
PS1 = 1;
PS2 = 1;
```

```
Firmware successfully loaded
```

```
-----------------------------------------------------------
File: D:/Project/RGS_DEV/EMBED_TRAINING/SimulCode/interrupt.c
```

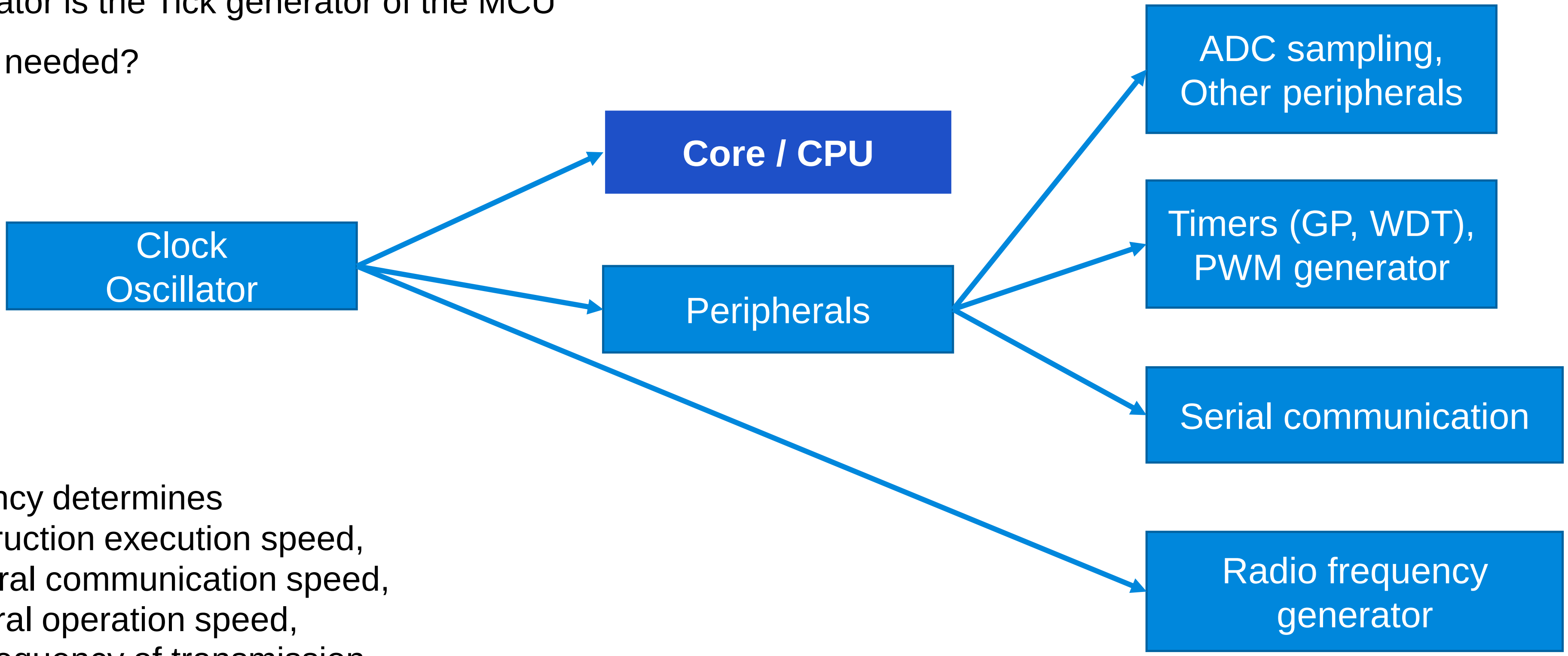
Summary

- GPIO works in Boolean signal
- SFRs are special registers include Data, status & Control
- The configuration of peripherals are done by writing to Control register
- C Program (the instructions), will be executed by the CORE
- The program tells the step by step instruction for the CORE, to configure the registers and to read / write data from / to the registers (SFRs)
- C Program can be written as 1 time execution set (configurations), and continuous execution set (Data in / out)

CLOCK

MCU Clock

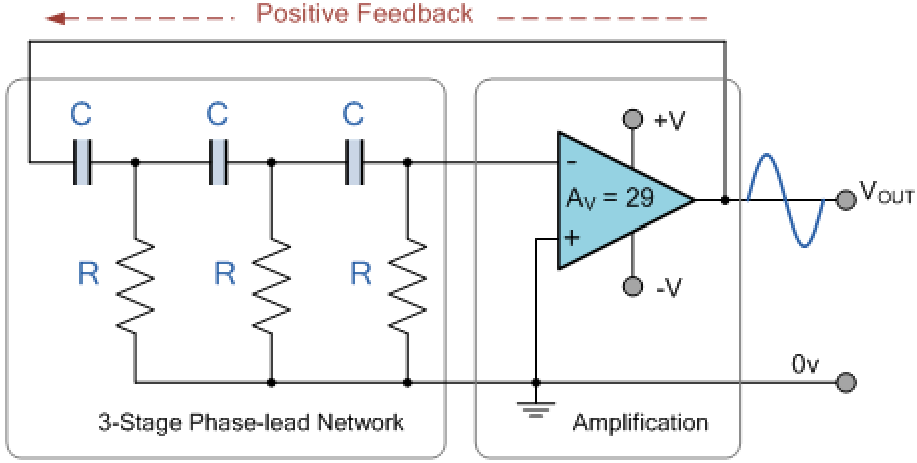
- Clock / Oscillator is the Tick generator of the MCU
- Why Clock is needed?

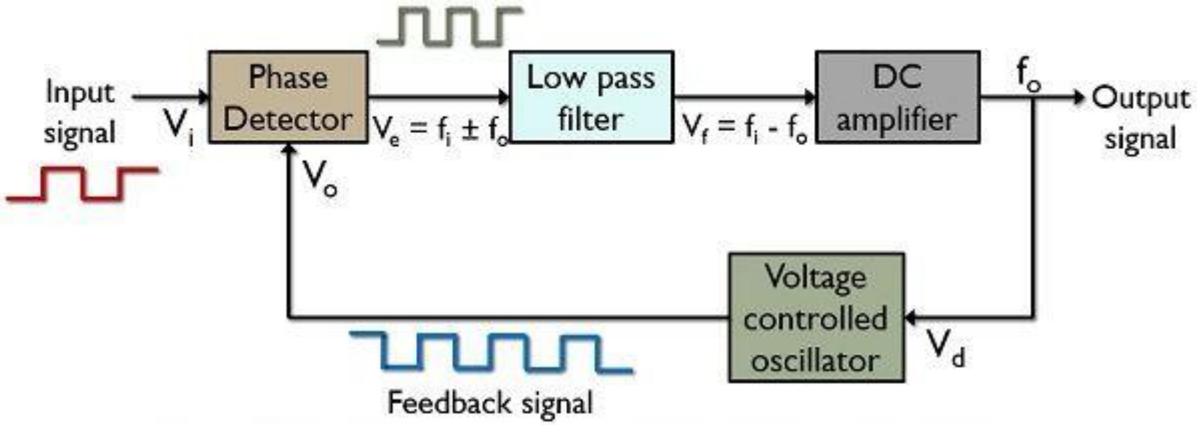

Clock Oscillator
Core / CPU
Peripherals
ADC sampling, Other peripherals
Timers (GP, WDT), PWM generator
Serial communication
Radio frequency generator

- Clock frequency determines
  - the Instruction execution speed,
  - Peripheral communication speed,
  - peripheral operation speed,
  - audio frequency of transmission
- Time synchronization and measured in Hz (MHz mostly)
- How Many clock cycles the CORE takes for an execution of Instruction
  - PIC MCU takes 4 cycles (decode, read, process, and write)

Types of Clock (oscillators)

- Crystal Oscillators
  - (TXCO, LP, XT, HS), Piezo electric effect
- RC oscillator
- Ceramic resonator
- External clock
- Internal oscillator
- PLL – Phase Locked Loops
- RTC clock (32.768kHz)



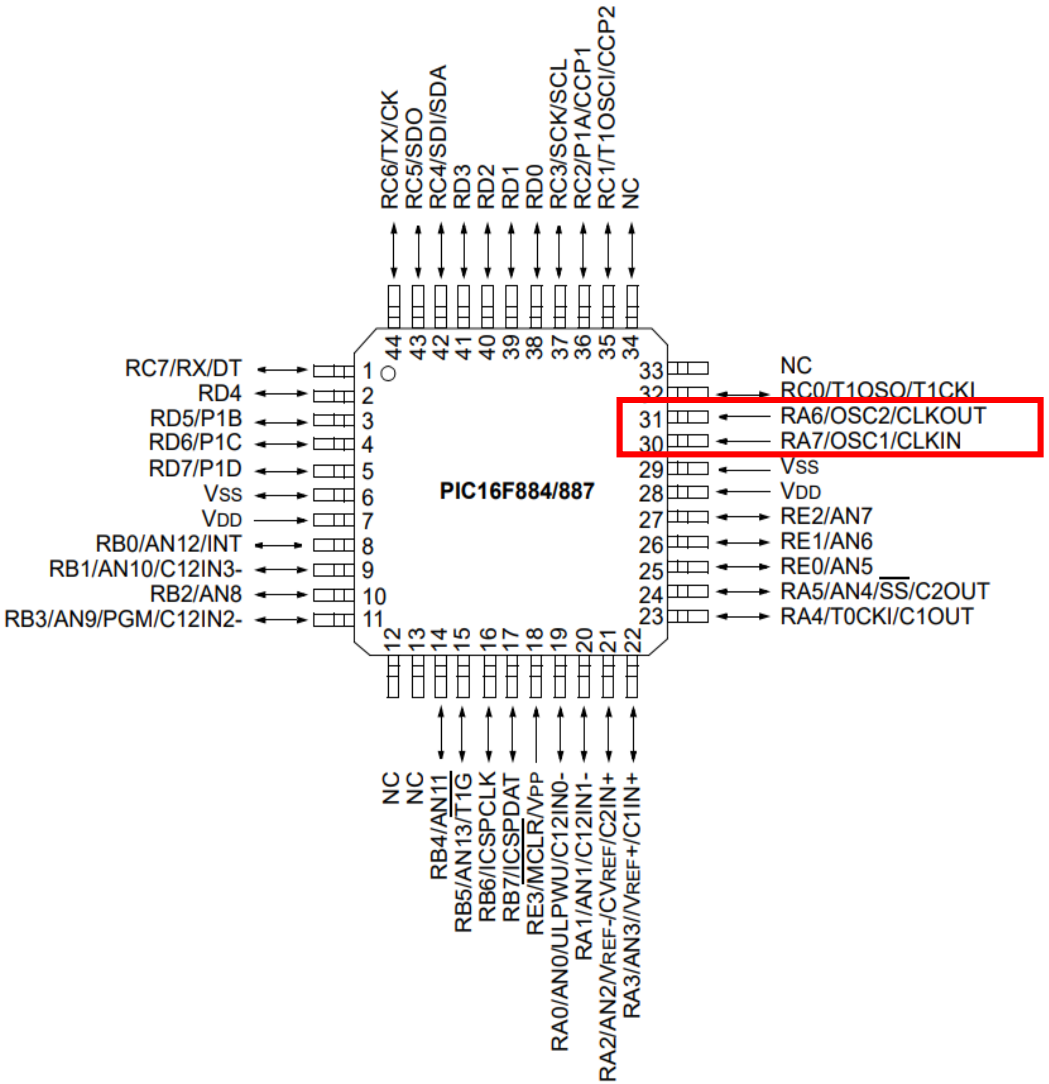


Block Diagram of Phase-Locked Loop

Electronics Coach

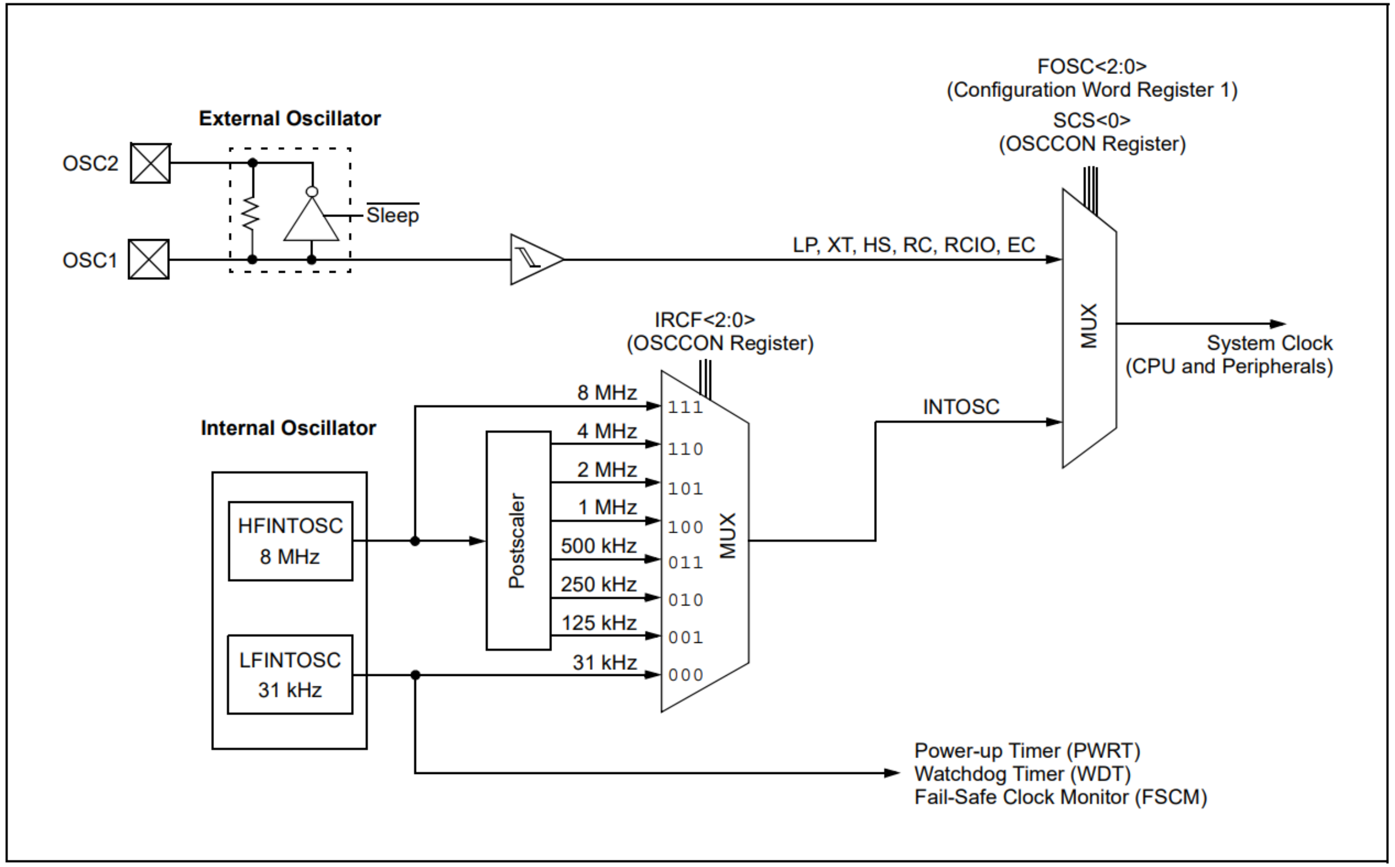
Configuration in PIC16F887

1. EC – External clock with I/O on OSC2/CLKOUT.
2. LP – 32 kHz Low-Power Crystal mode.
3. XT – Medium Gain Crystal or Ceramic Resonator Oscillator mode.
4. HS – High Gain Crystal or Ceramic Resonator mode.
5. RC – External Resistor-Capacitor (RC) with FOSC/4 output on OSC2/CLKOUT.
6. RCIO – External Resistor-Capacitor (RC) with I/O on OSC2/CLKOUT.
7. INTOSC – Internal oscillator with FOSC/4 output on OSC2 and I/O on OSC1/CLKIN.
8. INTOSCIO – Internal oscillator with I/O on OSC1/CLKIN and OSC2/CLKOUT.

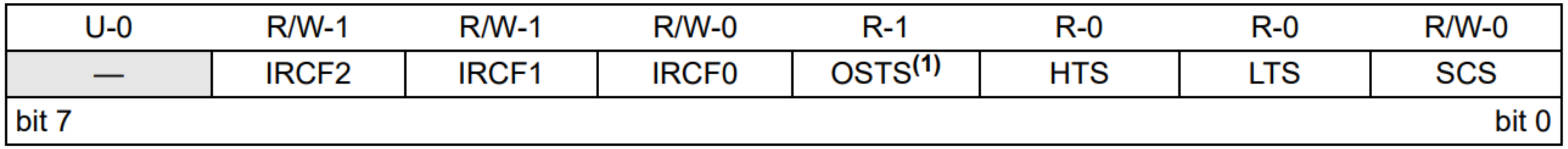


Configuration in PIC16F887

FIGURE 4-1: PIC® MCU CLOCK SOURCE BLOCK DIAGRAM

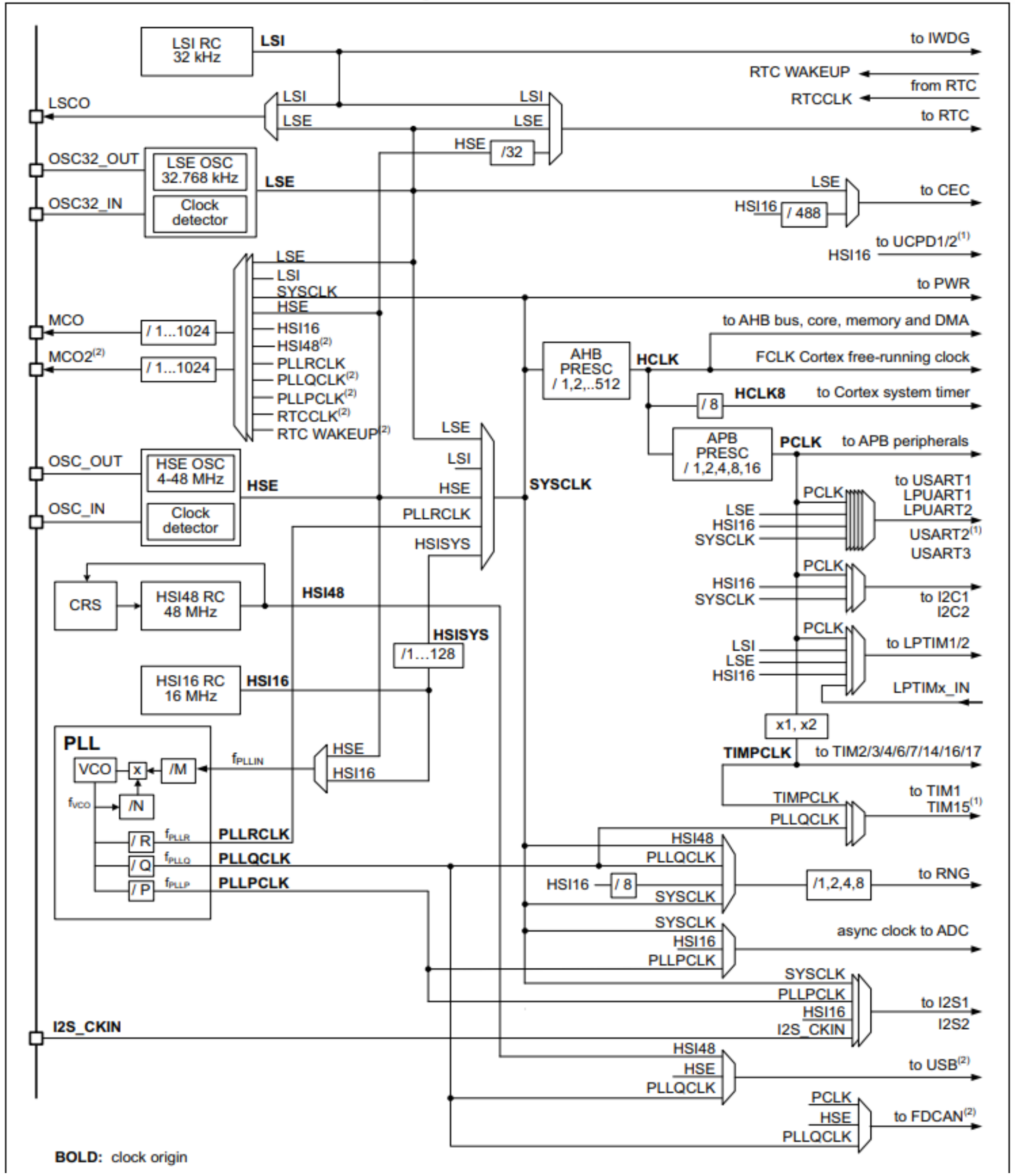


REGISTER 4-1: OSCCON: OSCILLATOR CONTROL REGISTER



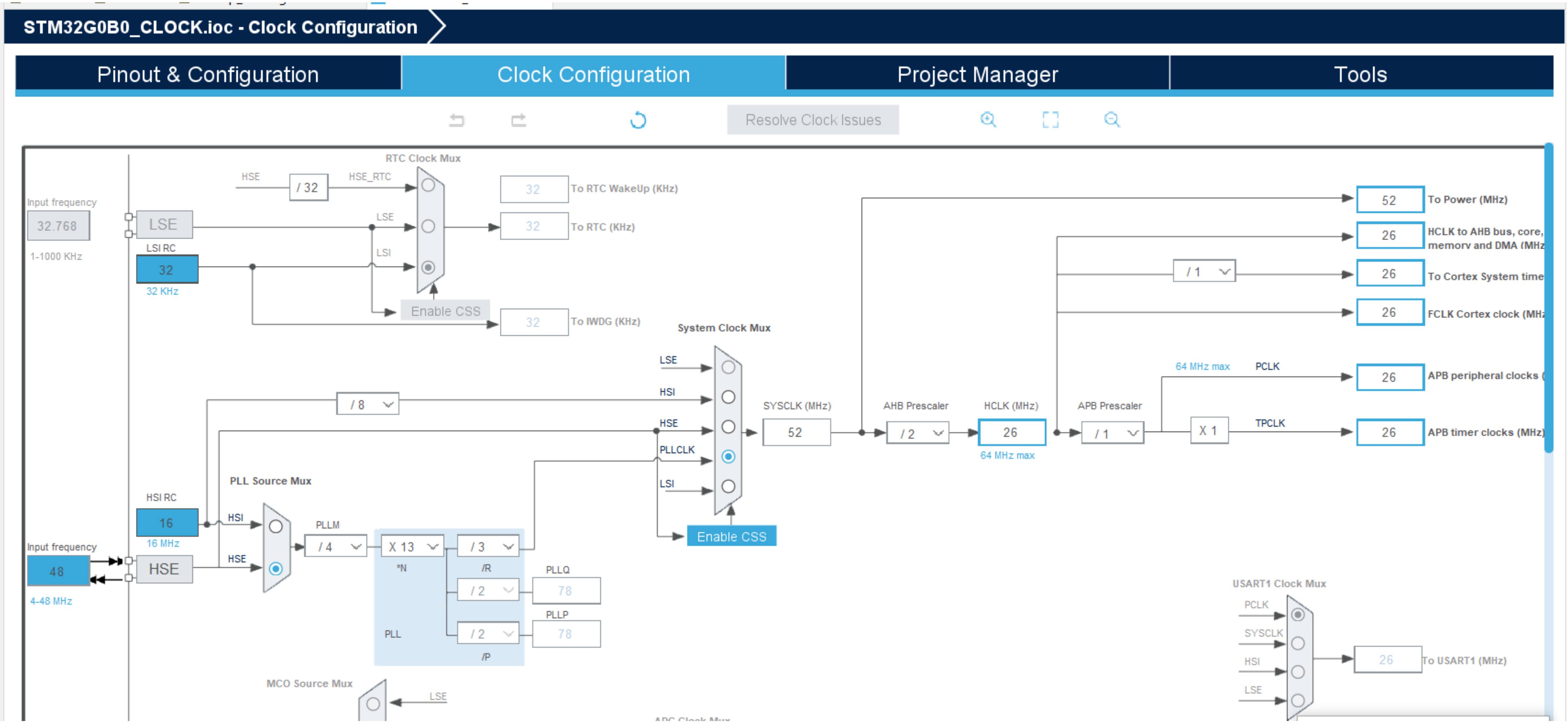
U-0	R/W-1	R/W-1	R/W-0	R-1	R-0	R-0	R/W-0
—	IRCF2	IRCF1	IRCF0	OSTS[1]	HTS	LTS	SCS
bit 7							bit 0

Clock in STM32G0B0CE



Refer Pg 124 in STM32G0x0 Reference manual

Hands on with Clock configurator



STM32G0B0_CLOCK.ioc - Clock Configuration

Pinout & Configuration | Clock Configuration | Project Manager | Tools

Resolve Clock Issues

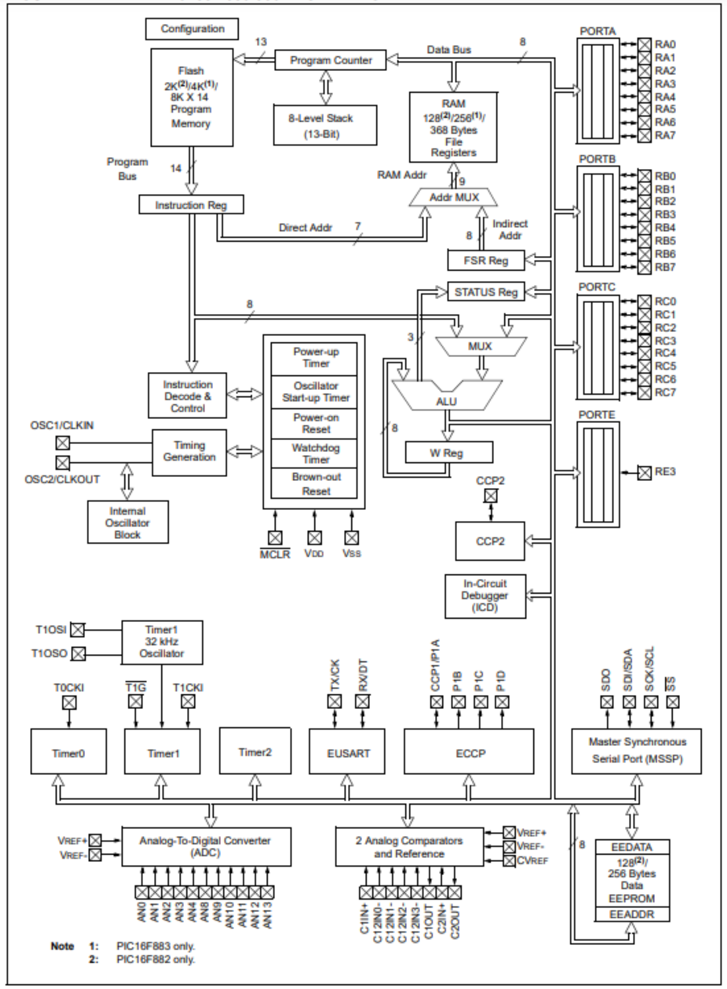
Summary

- Clocks internal / external, crystal, resonator, RC -> used for Core and peripheral operation
- PLL -> High speed clock generation
- Careful with OSC vs OSC_32 (RTC clock)

TIMER & COUNTER
(*Peripherals*)

Internal architecture

FIGURE 1-1: PIC16F882/883/886 BLOCK DIAGRAM



Note 1: PIC16F883 only.
2: PIC16F882 only.

DS41291D-page 14 Preliminary © 2007 Microchip Technology Inc.

Refer Pg No 16 in PIC16F887 datasheet

Refer Pg No 11 in STM32G0B0CE datasheet

Timer prerequisite discussion

- What is a timer?
- What the timer & counter peripheral do?
- Difference among Clock, timer, stopwatch, Alarm in Phone
- Why core cannot be used for the same timer actions?

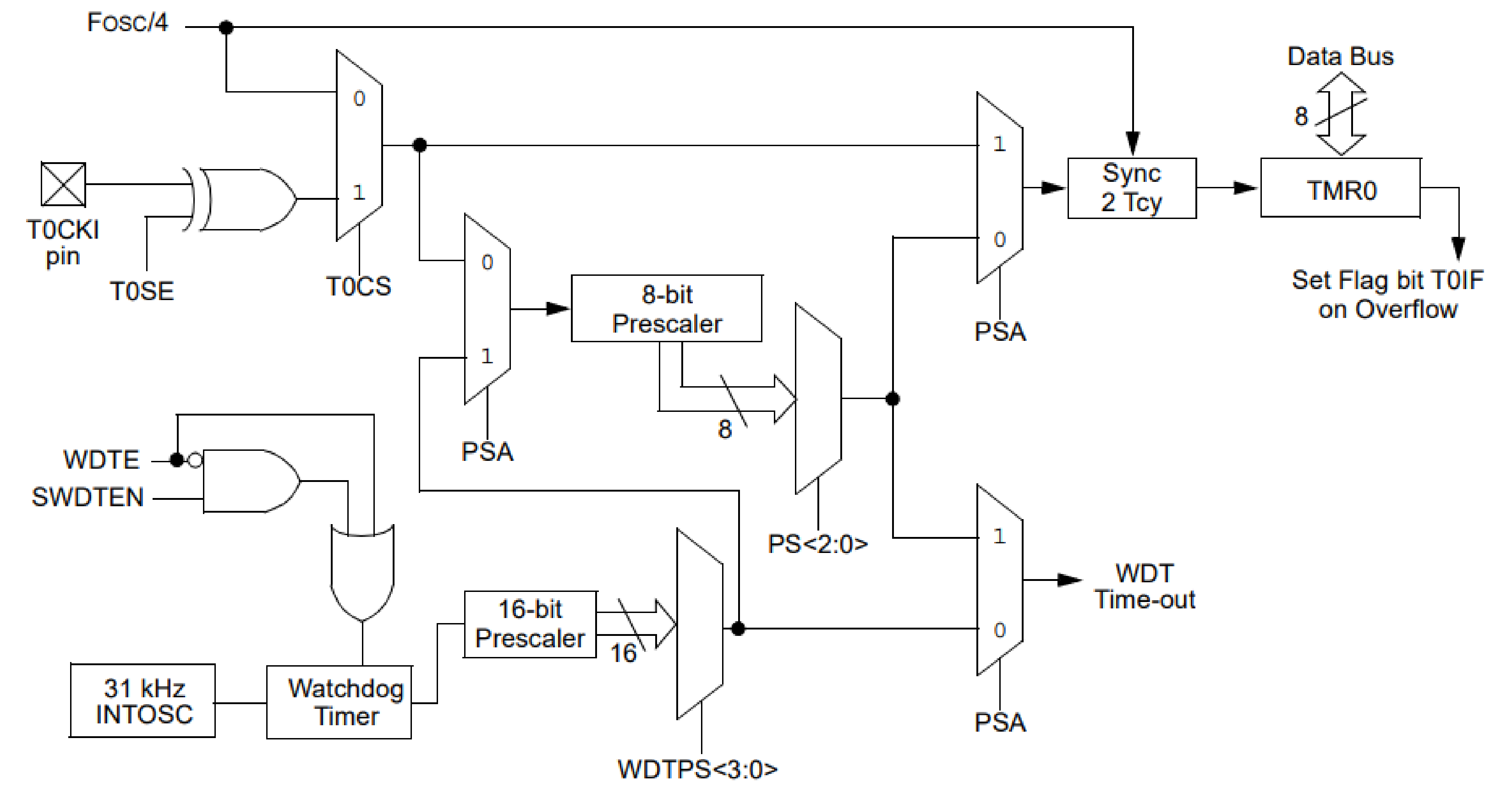
Timer concepts

- Types of Timers
  - General purpose timer

    - For general counting / timing purposes, in 8 bit, 16 bit / 32 bit
  - Watch dog timer
    - To watch the proper functioning of the system
  - RTC timer (real time clock)
    - To get real Human clock time
- In general purpose timer
  - Timers exist like Timer0, Timer1, Timer2 and may be sub-timers also A, B
  - *Pre scaler* – Increments 1 per *x* ticks for *1:x*
  - *Post scaler* – Generates interrupt after n overflows for 1:n
  - *Overflow* – Timer value changing from 0xFFFF to 0x0000
  - Up-counter vs downcounter, underflow

PIC16F887 Timer 0

FIGURE 5-1: TIMER0/WDT PRESCALER BLOCK DIAGRAM



Note 1: T0SE, T0CS, PSA, PS<2:0> are bits in the OPTION register.

2: SWDTEN and WDTPS<3:0> are bits in the WDTCON register.

3: WDTE bit is in the Configuration Word Register1.

PIC16F887 Timer 0 Registers

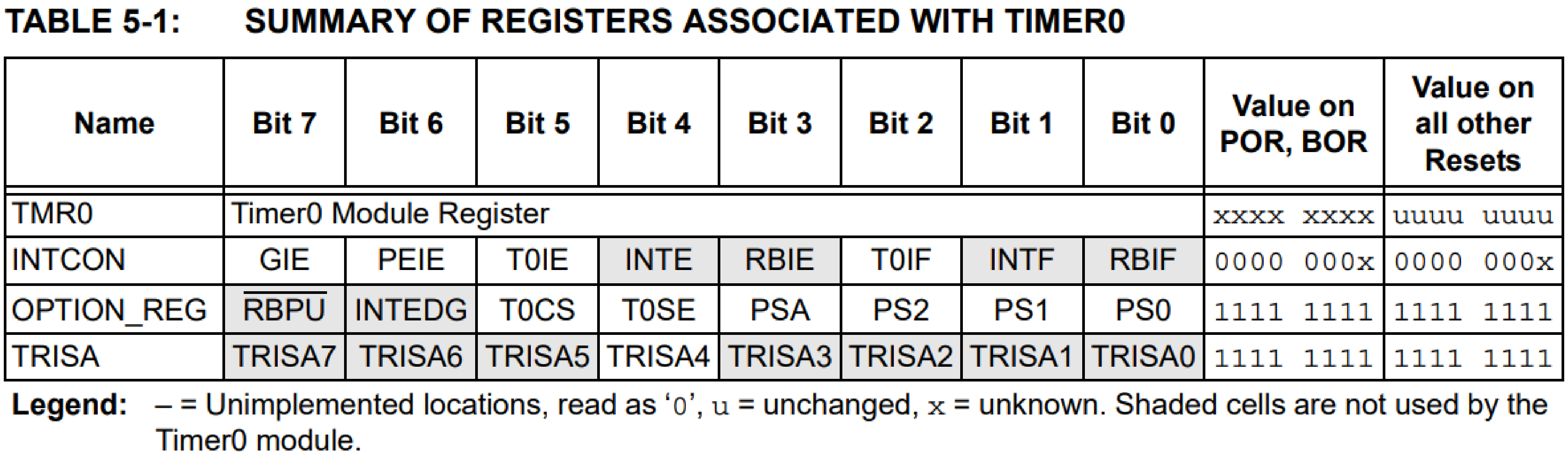


TABLE 5-1: SUMMARY OF REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
TMR0	Timer0 Module Register								`xxxx xxxx`	`uuuu uuuu`
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	`0000 000x`	`0000 000x`
OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	`1111 1111`	`1111 1111`
TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	`1111 1111`	`1111 1111`

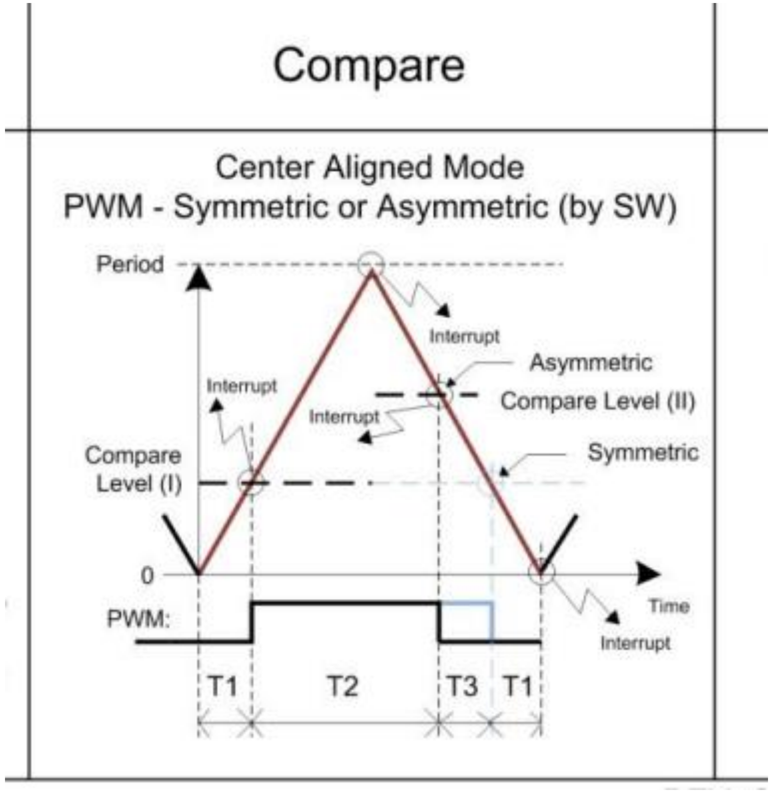
Legend: – = Unimplemented locations, read as ‘`0`’, `u` = unchanged, `x` = unknown. Shaded cells are not used by the Timer0 module.

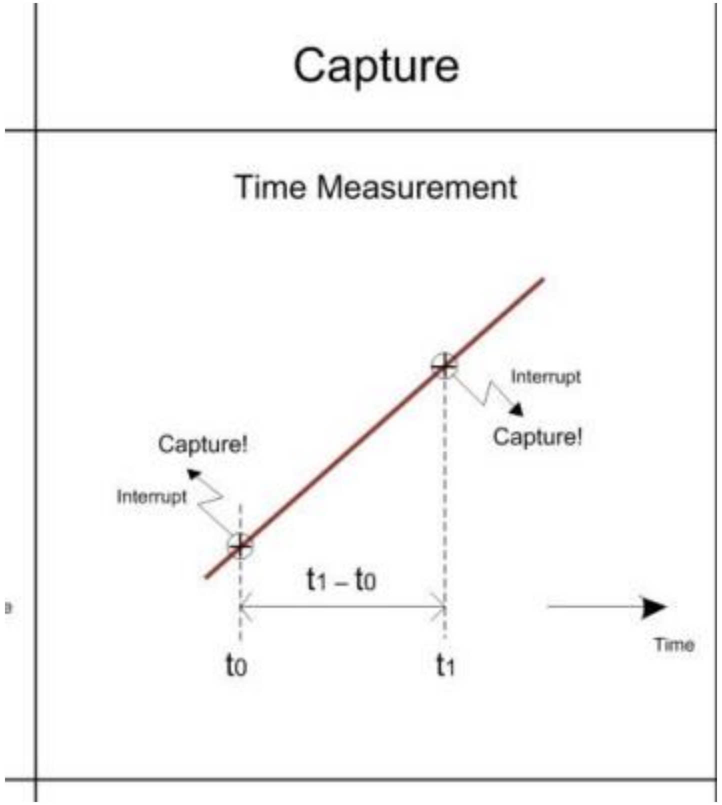
- Timer Calculations

Timer 1 & 2 in datasheet

CCP Module

- C – Compare
  - Compare any signal / value with Timer values, trigger events
- C – Capture
  - Capture the timer value between the intervals
- P – PWM



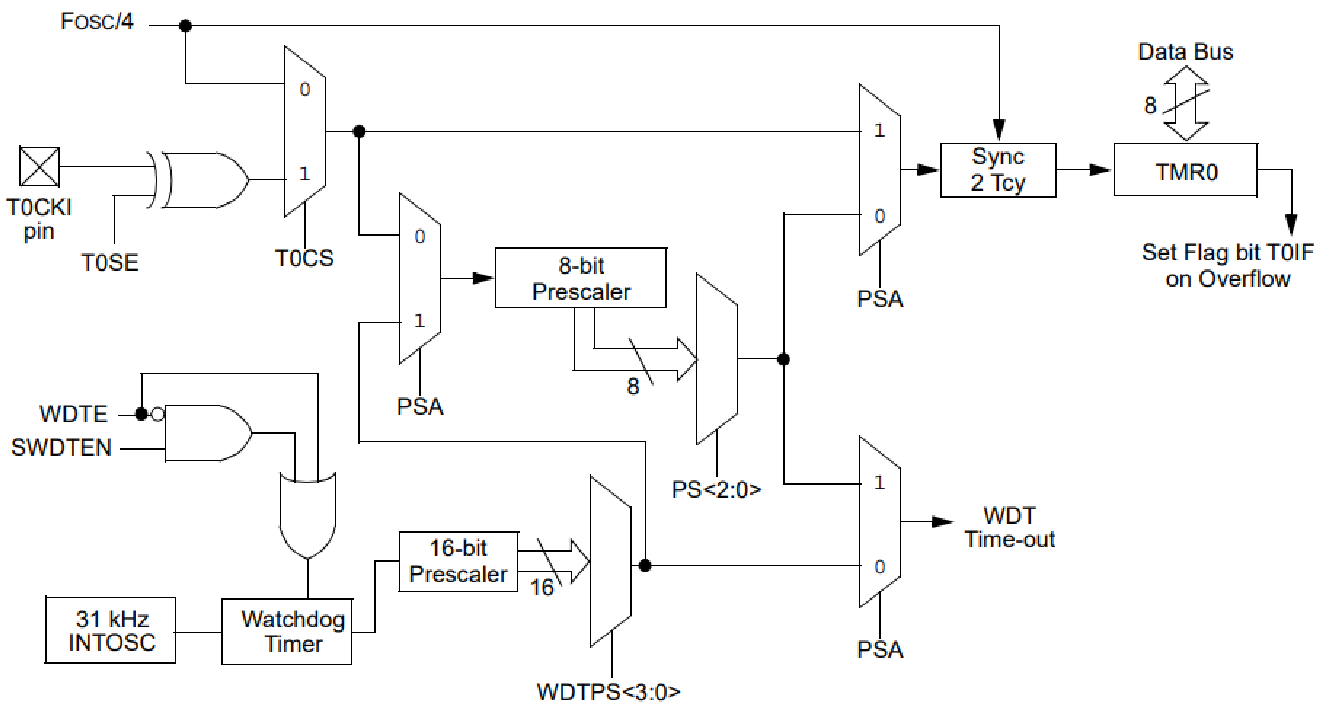


Timer 1 & 2 in datasheet

Counter

- Edge detection

FIGURE 5-1: TIMER0/WDT PRESCALER BLOCK DIAGRAM



Note 1: T0SE, T0CS, PSA, PS<2:0> are bits in the OPTION register.

2: SWDTEN and WDTPS<3:0> are bits in the WDTCON register.

3: WDTE bit is in the Configuration Word Register1.

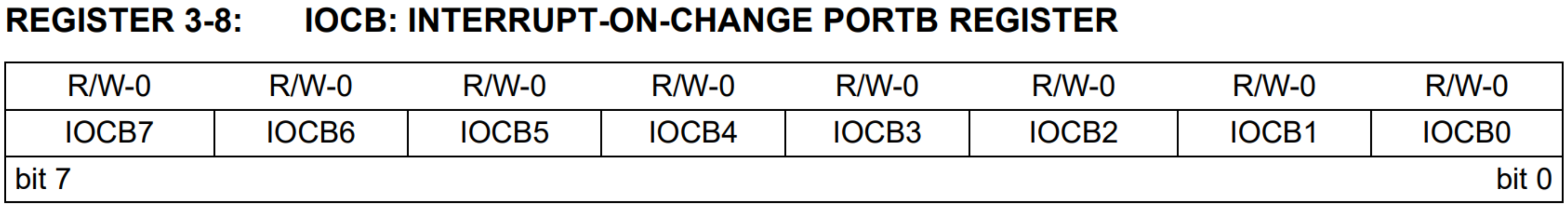
INTERRUPTS

Interrupt concepts

- Interrupts
  - Stops the execution of program, jumps control to ISR
- ISR
  - Interrupt service routine
- Interrupt Vector
  - Interrupt service routine
- Interrupt flags
  - Interrupt service routine
- Interrupt priorities
- Interrupt source
- Mask-able and non mask-able interrupt

Interrupt concepts

- When to use interrupts?
  - Always use when action done by external / peripheral
- How could be an ISR?
  - Very short as possible



REGISTER 3-8: IOCB: INTERRUPT-ON-CHANGE PORTB REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
bit 7							bit 0

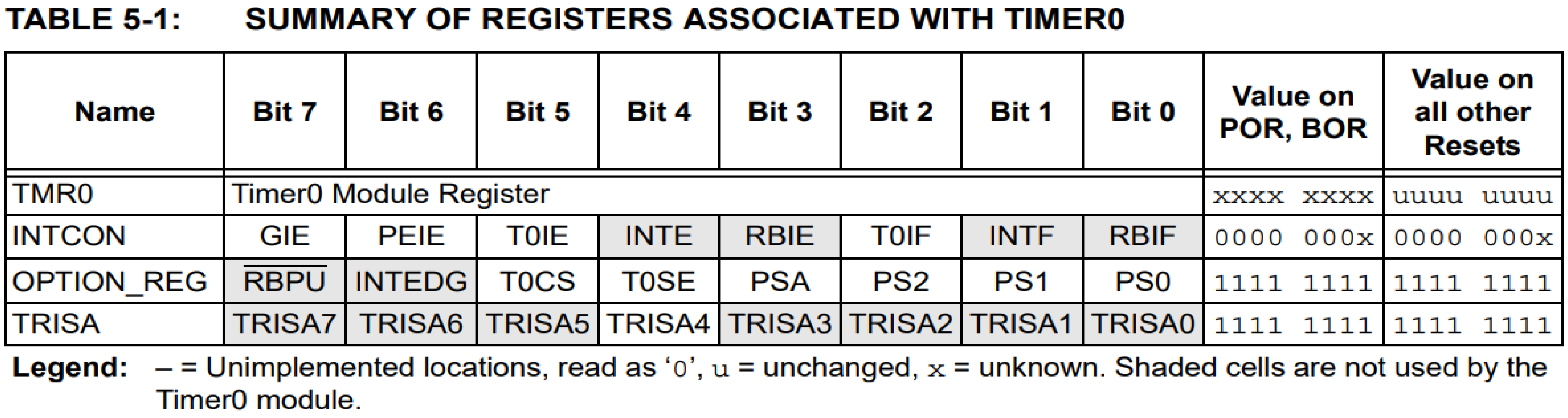


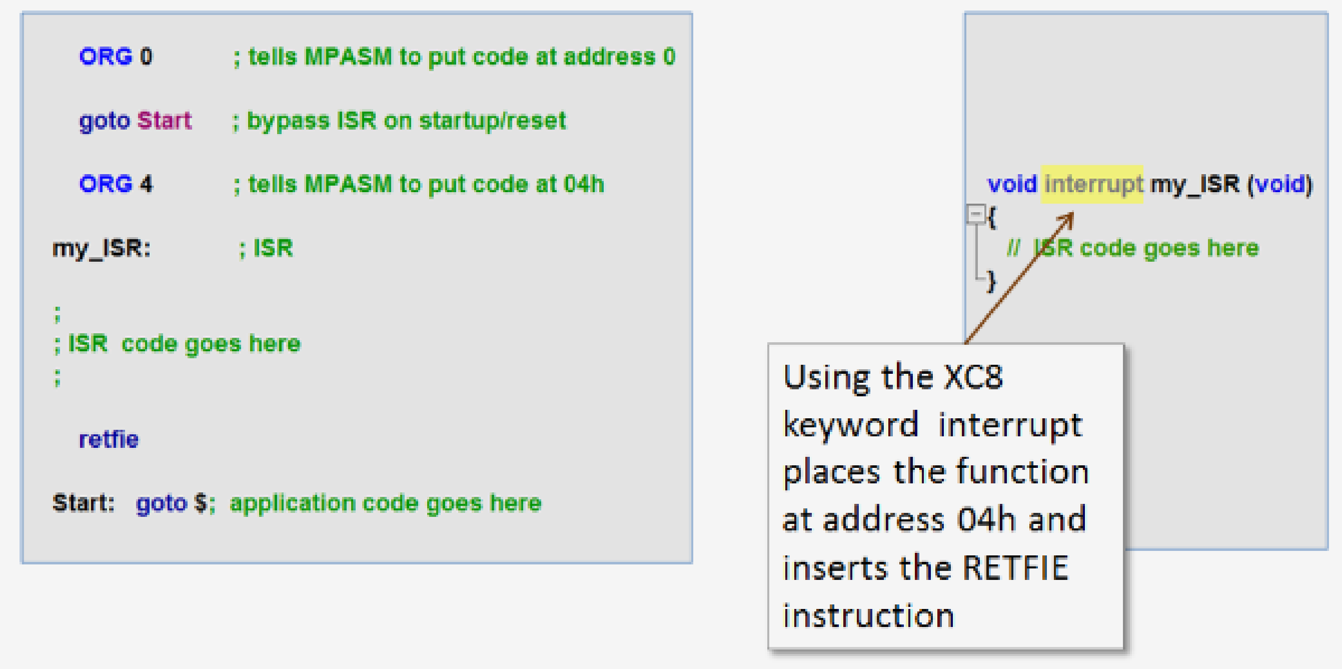
TABLE 5-1: SUMMARY OF REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x
OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	1111 1111

Legend: – = Unimplemented locations, read as '0', u = unchanged, x = unknown. Shaded cells are not used by the Timer0 module.

Interrupt Handlers and Vectors

The predefined interrupt handler function declared by xc8 compiler

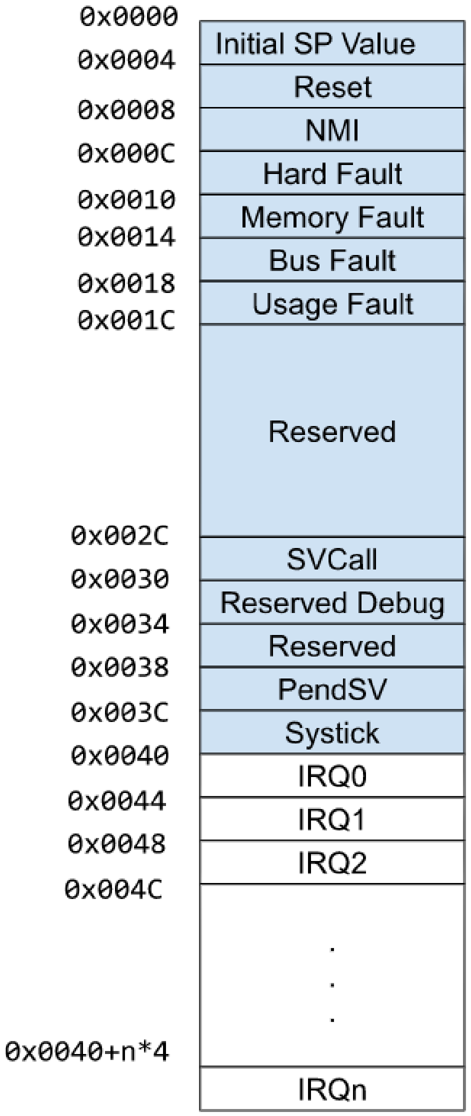


```
    ORG 0         ; tells MPASM to put code at address 0

    goto Start    ; bypass ISR on startup/reset

    ORG 4         ; tells MPASM to put code at 04h

my_ISR:            ; ISR

;
; ISR  code goes here
;

    retfie

Start:   goto $;  application code goes here
```

```
void interrupt my_ISR (void)
{
   // ISR code goes here
}
```

Using the XC8 keyword interrupt places the function at address 04h and inserts the RETFIE instruction

ARMs flash structure for Interrupt Vector



0x0000	Initial SP Value
0x0004	Reset
0x0008	NMI
0x000C	Hard Fault
0x0010	Memory Fault
0x0014	Bus Fault
0x0018	Usage Fault
0x001C	Reserved
0x002C	SVCall
0x0030	Reserved Debug
0x0034	Reserved
0x0038	PendSV
0x003C	Systick
0x0040	IRQ0
0x0044	IRQ1
0x0048	IRQ2
0x004C	. . .
0x0040+n*4	IRQn

Smarter. Greener. Together.

Thank you